Demystifying Incentives in the Consensus Computer

Loi Luu
School of Computing
National University of Singapore
loiluu@comp.nus.edu.sg

Jason Teutsch
School of Computing
National University of Singapore
teutsch@comp.nus.edu.sg

Raghav Kulkarni
Centre for Quantum Technologies
National University of Singapore
kulraghav@gmail.com

Prateek Saxena
School of Computing
National University of Singapore
prateeks@comp.nus.edu.sg

ABSTRACT

Cryptocurrencies like Bitcoin and the more recent Ethereum system allow users to specify scripts in transactions and contracts to support applications beyond simple cash transactions. In this work, we analyze the extent to which these systems can enforce the correct semantics of scripts. We show that when a script execution requires nontrivial computation effort, practical attacks exist which either waste miners' computational resources or lead miners to accept incorrect script results. These attacks drive miners to an ill-fated choice, which we call the *verifier's dilemma*, whereby rational miners are well-incentivized to accept unvalidated blockchains. We call the framework of computation through a scriptable cryptocurrency a consensus computer and develop a model that captures incentives for verifying computation in it. We propose a resolution to the verifier's dilemma which incentivizes correct execution of certain applications, including outsourced computation, where scripts require minimal time to verify. Finally we discuss two distinct, practical implementations of our consensus computer in real cryptocurrency networks like Ethereum.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; K.4.4 [**Computers And Society**]: Electronic Commerce—*Cybercash, digital cash*

Keywords

Bitcoin; Ethereum; cryptocurrency; incentive compatibility; verifiable computation; consensus computer

1. INTRODUCTION

Cryptocurrencies such as Bitcoin [1] are attracting massive investments in computing power, and the power consumed has been growing exponentially in recent years [2]. Bitcoin can be viewed as a large network of miners competing in a lottery that awards newly minted currency, called *Bitcoins*, in exchange for contributing computational resources to solutions of cryptographic puzzles, or *blocks*. Bitcoin miners collectively agree upon who receives the minted Bitcoins and which *transactions* to accept. This process of *consensus*, or agreement by majority, permanently records decisions in a public ledger called the *blockchain*. More than 50 cryptocurrencies use similar blockchain protocols. While the core blockchain mechanism has been used for establishing a public ledger of who-pays-whom transactions, it has features that go beyond this function. Specifically, the blockchain supports a light-weight *scripting* language, designed primarily to allow conditional transactions which can be repurposed for other applications. Emerging cryptocurrencies can now enable computation for applications such as financial back-offices, prediction markets, distributed computation (*e.g.*,, Gridcoin for BOINC [3]), and perhaps even a decentralized Linux OS [4].

In Bitcoin, a transaction defines a particular activity in the network, *e.g.*, sending Bitcoin between users. Transactions may include a script that specifies a validity condition. Figure 1 illustrates a basic transaction having a script check whether the payee in the transaction has the private key corresponding to the recipient's Bitcoin wallet address. More interestingly, next-generation cryptocurrencies such as Ethereum [5] introduce a Turing-complete script language which allows users to encode arbitrary computation as scripts and support a variety decentralized applications. The large number of miners on the cryptocurrency network, who both execute and verify computational tasks, reach agreement through an established consensus protocol. We therefore refer collectively to these miners as *verifers*, and the computation framework of scriptable cryptocurrencies as a *consensus computer*. We wish to characterize the classes of computation that users can trust a cryptocurrency network to execute and verify correctly.

Miners have two separate functions in the consensus computer: checking that blocks are correctly constructed, or *proof-of-work*, and checking the validity of transactions in each block. While verifying correct block construction requires a relatively small amount of work (two `SHA256` calculations), checking the validity of transactions contained in a block can take much more time for two reasons. First, the number of transactions per block may be large (≈ 800 in Bitcoin at the time of writing, and its capacity may soon

CCS'15, October 12–16, 2015, Denver, Colorado, USA.
© 2015 ACM. ISBN 978-1-4503-3832-5/15/10$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813659.

be extended to support high transaction rates [6, 7]). Second, expressive transaction scripts in emerging cryptocurrencies such as Ethereum can require significant computational work to verify. These expressions create a new dilemma for miners — whether the miners should verify the validity of scripted transactions or accept them without verification. Miners are incentivized to verify all scripted transactions for the "common good" of the cryptocurrency so to speak. However, verifying scripts consumes computation resources and therefore delays honest miners in the race to mine the next block. We argue that this dilemma leaves open the possibility of attacks which result in unverified transactions on the blockchain. This means that some computation tasks outsourced to cryptocurrency-based consensus computers may not execute correctly.

Our work makes three new contributions. First, we describe the *verifier's dilemma* in emerging cryptocurrencies which shows that honest miners have an ill-fated choice: whether to validate a block's transactions or not. In either case, they are susceptible to a set of attacks from dishonest miners. We show that malicious miners can attack their peers with zero financial risk via the scriptability feature of cryptocurrencies. Furthermore, our verifier's dilemma implies that rational miners have incentive to skip the verification of expensive transactions to gain a competitive advantage in the race for the next block. However, this results in an unvalidated blockchain containing unverified computation results.

Second, we propose a security model to formalize the consensus computer. Our model allows us to study the incentive structure and attacks that affect the correctness of computations performed on a consensus computer. Verifiable computation methods on consensus computers differ from techniques that have been used on classical computers [8, 9, 10, 11, 12, 13, 14]. A consensus computer allows complete decentralization of verification — the puzzle giver need not trust any individual verifiers on the network or the prover who provides the solution. In our model, the network is assumed to implicitly agree on correct transactions if and only if the incentives donot advantage dishonest miners. Verifiable computation techniques for classical setting have a different goal: that of producing an explicit cryptographic proof of the correctness of the computation. Often, such techniques require an involved key setup phase, and have impractical computational overheads for the prover.

Previous works implicitly assume that the Bitcoin consensus computer will always generate correct solutions, *i.e.*, miners will verify and agree on correct transactions [15, 16, 17, 18, 19, 20, 21]. Our present model provides a formal explanation as to why this assumption holds and suggests potential constraints of consensus computation in cryptocurrencies with more expressive scripting languages. Specifically, when the computational advantage of skipping verification is low, say ε, rational miners gain little by cheating thus behaving honestly to give correct solutions. We call a system restricted to such primitives an *ε-consensus computer* and expect it to compute correctly.

We propose two mechanisms to realize an ε-consensus computer on Ethereum. Our first approach allows us to achieve correctness by splitting the computation into several smaller steps such that the consensus correctly verifies each step. This approach achieves exact correctness in results but with higher computational burden to the network.

```
Input:
  PreviousTX: ID of previous transaction
  Index: 0
  scriptSig: Sign(PubKey), PubKey

Output:
  Value: 5000000000
  scriptPubKey: %take Signature and PubKey as params
    checkif Hash(PubKey) = Payee's ID,
    checkif Sign(PubKey) is valid
```

Figure 1: Illustration of a simple transaction in Bitcoin. User's address is computed by hashing their public key. `scriptPubKey` is the script that defines how the payee claims the recieved Bitcoin.

Our second mechanism allows for *approximate correctness*. Specifically, we allow the approximation gap between submitted and correct results to be tunable to a negligible quantity, at much lower computational cost. Whether one can design a distributed, cryptocurrency system which permits secure execution of a larger class of computations remains an interesting open problem, as is the problem of determining the class of puzzles whose solutions admit light-weight verification.

Contributions. In summary, our work makes the following contributions:

- *Verifier's dilemma and attacks.* We introduce a dilemma in which miners are vulnerable to attacks regardless of whether they verify a transaction or not. We further show that miners are incentivized to skip the verification and perform an attack to get more advantage in mining the next blocks.
- *Security model for a Consensus computer.* We formalize the computation and verification by a consensus computer. We investigate the incentive structure, threat model and conditions under which a consensus computer can realize correct outsourced computation.
- *Techniques to realize an ε-consensus computer.* We propose techniques to realize our ε-consensus computer in real cryptocurrency networks like Ethereum. We illustrate the practical utility of our techniques with examples from outsourced computation.

2. BACKGROUND: CRYPTOCURRENCIES

2.1 The consensus protocol

Most cryptocurrencies use a public peer-to-peer consensus protocol known as *Nakamoto consensus*, named after and introduced by the founder of Bitcoin [1], which does not require a central authority. At the heart of this protocol is a blockchain which acts as a public ledger and stores the complete transaction history of the network. The security of the blockchain is maintained by a cryptographic chain of puzzles (or blocks). Miners validate and approve transactions while generating, or *mining*, new blocks. Mining a new block rewards newly minted coins to one of the miners that demonstrates by consensus that it successfully solved a designated cryptographic puzzle. Figure 2 concisely illustrates the structure of the blockchain data structure.

The protocol uses consensus in two places to make the cryptocurrency robust. First, the network must agree on the rules to verify valid blocks and transactions. Second,

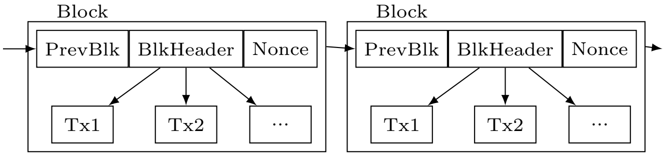

Figure 2: The Blockchain in popular cryptocurrencies like Bitcoin and Ethereum. Each block consists of several transactions (Txs).

the data in the blockchain must be consistent across miners, so that everyone knows who owns what. Thus the blockchain acts as a base to verify which transactions are valid.

2.2 Transactions & scriptability

A transaction in a cryptocurrency defines a particular activity in the network of that currency. For example, Figure 1 is a basic transaction in Bitcoin which transfers Bitcoins from sender to receiver. The `scriptPubKey` component (Line 8) allows the sender to define the receiver's address and on which condition he can spend the Bitcoins. The sender can dynamically program the `scriptPubkey` to support various use cases.

Transaction verification. In order to verify a transaction, one has to check whether the input provided in `scriptSig` satisfies the logic encoded in the `scritptPubkey` of the `PriviousTX`. For example, in Figure 1, miners will check if the receiver is the intended payee (on Line 9) and he indeed owns the payee's address (on Line 10). The Bitcoin protocol states that the verification of a transaction `TX` should happen in two places, when:

- *A new transaction is broadcast* (step 1). When a user broadcasts `TX` to a miner, the miner verifies if the transaction is correct according to the latest blockchain state. If so, he includes it in his block header to mine a new block and propagates the transaction to his neighbors.

- *A new block is broadcast* (step 2). When `TX` is included in a newly found block, before accepting the block everyone will check the correctness of all transactions contained in the block.

2.3 Incentivizing correctness

While it is clear that Nakamoto consensus incentivizes a miner to mine new blocks with a reward of newly minted coins, the incentive for others to correctly verify the transactions is not generally understood. The folklore reasons for why miners verify transactions in the Bitcoin blockchain are:

- Verifying a transaction at both steps requires negligible additional work compared to mining a new block. For example, the Bitcoin transaction in Figure 1 has only 4 opcodes. Miners can do this extra check without expending any significant computational work. The cost of validation outweighs the miner's risk of having their earned Bitcoins in new blocks be discarded, if the invalidity is detected in the future.

- When receiving a new block, miners can accept it with or without verifying the included transactions. However, *most* miners want to maintain a cheap and correct system (the "common good"), so that the Bitcoin blockchain remains healthy and Bitcoins act as a store-of-value. Thus miners check the validity of a block's included transactions for free [1].

```
code:
    if msg.datasize==2:
        return msg.data[0] + msg.data[1]
```

Figure 3: An Ethereum contract that returns sum of two numbers.

In this work, we study the financial incentives of users more carefully to understand when these folklore assumptions may fail and what happens when they do.

2.4 Ethereum—Turing-complete scripting

Ethereum is a next generation cryptocurrency which enables more applications beyond those supported by Bitcoin [5]. It provides a Turing-complete programming language in its design which equips users with a mechanism to express concrete conditional semantics for transactions. Ethereum and Bitcoin share nearly identical incentive structures. The only conceptual difference is that Ethereum includes a small reward for late, valid "uncle block" [5]. However, as in Bitcoin, Ethereum miners maximize their rewards by racing to extend the longest valid block. Ethereum also introduces *smart contracts* which permit many potential applications to run on top of the blockchain.

Smart contract. A *smart contract* is an entity on the Ethereum blockchain which can embed many contractual clauses and make it expensive for anyone to disoblige or deviate from the contract after agreement [22]. Each smart contract has its own address, balance, and storage space which is used to run a specified script. While Bitcoin only allows users to encode a stateless program, smart contracts support stateful programs. Users can trigger a contract by sending a transaction to its address. Once a smart contract transaction gets included in the blockchain, everyone in the network is expected to execute the contract script to verify its validity. Figure 3 is a simple contract which returns the sum of two numbers.

Gas system. It may be obvious to some readers that having Turing-complete language in Ethereum script is a problem. More specifically, users can write a transaction or contract script with long verification time to perform Denial-of-Service (DoS) attack on the network. For example, one can write a simple transaction to make the network enter an infinite loop while verifying the transaction. To prevent such attacks, Ethereum introduces the concept of *gas* [23]. The script is compiled into Ethereum opcodes while stored in the blockchain, and each opcode costs some predefined amount of *gas* (a form of transaction fee) charged to transaction sender. When a sender sends a transaction to activate a contract, he has to specify the available `gasLimit` that he supports for the execution. The gas is paid to a miner who verifies and includes the transactions in his block. Intuitively, the gas system seems to make it expensive for the attacker to perform DoS attack to the system. However, as

[1] Note that only the one who finds the block receives a transaction fee, not those who verify.

```
1 init:
2     #Record the initiator, reward and data
3     contract.storage[0]=msg.sender
4     contract.storage[1]=0
5     contract.storage[2]=msg.value
6     #record the size n of the matrices
7     contract.storage[3]=msg.data[0]
8     #record the matrices A and B
9     contract.storage[4] = msg.data[1]
10    contract.storage[5] = msg.data[2]
11    contract.storage[6] = 1 #status
12 code:
13    if contract.storage[6] == 0:
14        return (1)
15    #data[C]: C is the result matrix
16    if msg.datasize == 1:
17        C = msg.data[0]
18        n = contract.storage[3]
19        A = contract.storage[4]
20        B = contract.storage[5]
21        # checking the result
22        for i in range(n):
23            for j in range(n):
24                cell = sum([A[i][k] * B[k][j]
25                    for k in range(n)])
26                if cell != C[i][j]:
27                    return (0)
28        #if everything is fine, send the reward
29        send(1000,msg.sender,contract.storage[2])
30        contract.storage[6]=0 #update status
31        contract.storage[7]=C #store result
32        return (2)
```

Figure 4: Code snippet of a contract which allows an user can outsource a matrix multiplication problem. The contract will verify the correctness of the result before sending out the reward.

we later show, this mechanism does not actually solve the DoS attack problem.

3. THE VERIFIER'S DILEMMA

In Section 2.3, we discussed incentives for Bitcoin miners to verify transactions in a block. We show that these motivations fail when block verification requires significant computation effort. Various mechanisms can lead to longer verification times. Ethereum users can create transaction and contract scripts which place arbitrary computational demands on miners, and similarly Bitcoin miners face high computational demands when the number of transactions in a single block is large.

In this section we present a *verifier's dilemma* in which the honest miners in the network decide whether to skip the verification of expensive transactions or to maintain the common good. To describe the verifier's dilemma, we first introduce a motivating smart contract in Ethereum which we will use throughout the paper.

3.1 Example of outsourced computation

Figure 4 shows a code snippet of a contract in Ethereum which allows a problem giver (G) to ask anyone to compute $A \times B$ where A, B are two matrices of size $n \times n$. In the `init` phase, G first sends an transaction which specifies n, A, B, and deposits the reward amount to the contract's wallet address. All information is public including the reward amount since everything is stored on the blockchain. If a prover (P), *i.e.*, a user is interested in solving the problem, wants to claim the solution to get the reward, he sends a transaction with the result matrix C to the verifier.

When other miners receive the transaction from the prover, they will verify if $C = A \times B$ by running the `code` phase in the contract script. G expects that the Nakamoto consensus protocol used in Ethereum will ensure that the result is correct due to the check on Lines 22–27, and only one prover will get paid due to the update of the contract's status on Line 30. Note that for the purpose of demonstration, we use an example in which verifying C requires to run the actual computation again. In practice, there are several problems that verifying whether a solution is correct is much easier than finding one, *e.g.*, solving a SAT instance, finding a hash inversion, breaking cryptography key and so on. Thus, G can create a contract to allow anyone to submit their solution and rely on the network to verify the correctness as in Figure 4.

3.2 Attacks

When a transaction that asks miners to verify whether $C = A \times B$ appears in the network, miners have an option to either verify or not to verify. We show that the miners are susceptible to a *resource exhaustion attack* or a *incorrect transaction attack* depending on their choice.

Attack 1 (Resource exhaustion attack by problem givers). If miners honestly follow the protocol, they voluntarily verify all transactions in a new block which is broadcast to them (step 2 in Section 2.2). Thus, if an attacker broadcasts his expensive transactions to the network, other miners will have to spend a lot of power and time to verify all the transactions. To prevent such situations, Ethereum introduced a *gas* system to charge the transaction initiator (sender) some amount of money for each operation he wants verified.

However, the gas system does not prevent the attacker from creating and including resource-intensive transactions in his newly minted block. This is because transaction fee (*i.e.*, gas) is collected by the block founder only (in step 1, Section 2.2). Thus, the attacker does not lose anything by adding his transactions to his own block. The other miners, on the other hand, have to spend a *significant* amount of time verifying those transactions (in step 2, Section 2.2) and get nothing in return. As a consequence, the attack not only exhausts other miners' resource, but also gives the attacker some time ahead of other miners in the race for the next block. Note that the attacker has complete freedom to prepare the transactions, so the difficulty of verifying the transaction script is in his control. Since gas charged in each transaction is credited to his account in step 1, his attack works at a zero fee (no loss).

As a concrete attack on the running example, the attacker first introduces the contract in Figure 4 with big matrices size, say $n = 1000$, and a small reward so that no one will attempt to solve it. In the second step, he includes a transaction with an arbitrary matrix C that he knows whether $C = A \times B$ before hand in all the blocks that he is mining. He also prepares enough gas to execute the contract script so that the transaction looks valid and normal. Other honest miners on receiving the block from the attacker spend time verifying the block and all the included transactions to see if the block is valid and move on to the next one in the chain. Since n is quite large, verifying a single transaction from the attacker will take significantly more time than normal transaction. The mining process of other miner will be delayed, while the attacker enjoys considerable advantage and has higher chance in finding the next valid blocks in the blockchain.

Remark 1. In Ethereum, the number of operations one includes in a new block is bounded by `oplimit`, which can be varied by a certain rate after every block by miners [5]. However, we argue that `oplimit` does not completely prevent the above attack since attackers can increase `oplimit` to a large enough value after some blocks to include his resource-intensive transactions. We explain this in the Appendix A. Further, some may argue that attacker's blocks are likely to become stale, *i.e.*, other blocks get broadcast to the network faster, due to the long verification time. However, in [24], Miller *et al.* find that there are (approx. 100) influential nodes in the Bitcoin network which are more efficient in broadcasting blocks. An attacker can broadcast his block to those influential nodes to reduce the chance that his blocks getting staled *significantly*.

Attack 2 (Incorrect transaction attack by provers). Due to Attack 1, rational miners have strong incentive to skip expensive transactions to compete for the race of the next block. The mechanism that drives the Bitcoin network (Section 2.3) to achieve a consensus does not hold in Ethereum. This is because verifying a transaction now may take much more time and affect the miners' mining speed. As a result, when the puzzle giver G asks for the product $A \times B$, the malicious prover P can include a transaction which has a wrong solution C. Since verifying $C = A \times B$ requires long time, rational miners are well-incentivized to accept it as correct without running the verification check. Thus, the result of a contract/transaction, although is derived from the consensus of the network, is incorrect and unreliable. The problem giver G wastes his money for an incorrect answer. Unlike in Attack 1, G is a honest miner in this case. However his computational power is not enough to match the rational miners who are incentivized to skip the check.

It is clear that skipping verification is problematic. First, P can include anything as the result of the contract execution, *e.g.*, sending others' money deposited in the contract's wallet to his wallet. Second, the health of the currency is affected since it is impossible to correctly verify who-owns-what. However, if the miners naïvely verify all the transactions, they are vulnerable to Attack 1.

3.2.1 Findings

Attack 1 and Attack 2 are not only specific to the running example, but are common challenges to any application that relies on a consensus protocol. From the *verifier's dilemma*, we establish the following findings and implications.

Claim 1 (Resource exhaustion attack)**.** In cryptocurrencies that allow miners to create expensive blocks, the honest miners are vulnerable to amplified resource exhaustion attacks. Malicious miners can perform the attack without fee and gain significant advantage in finding the next blocks.

The attack is applicable to most cryptocurrency once they become widely adopted, regardless of the applications running on top of the cryptocurrency. One can imagine the same problem will ocur in Bitcoin if block sizes were to increase dramatically, say to 1 GB per block. An attacker has incentive to create such a huge block to waste other miners' resources so as to gain advantage in the race for the next block. Such attacks have been reported in Bitcoin recently [25, 26]. It is shown in [26] that one can create a block-size transaction which requires miners to hash 19.1 GB of data and takes an average of CPU 3 minutes to verify. Bitcoin patched this vulnerability by allowing only pre-defined standard transactions, and thus limiting the potential applications of Bitcoin. Ethereum, on the other hand, has no such restrictions and permits users to define arbitrarily contracts.

Claim 2 (Nakamoto consensus permits unverified blocks)**.** Assuming all miners are rational, Nakamoto consensus may elect blockchains with unverified transactions.

Rationale. Miners in Nakamoto consensus-based cryptocurrenices are vulnerable to a resource exhaustion attack (Attack 1). The time and computational resource required to verify all transactions in a block expands significantly with number of transactions. By skiping verifications, a miner starts the race to find the next block earlier than honest miners. As a result, rational miners yield a longer blockchain by not verifying transactions in all blocks that they receive. By Nakamoto consensus, the longer chain will be considered the main chain [2] and contain unverified transactions. □

Incentive incompatibility in existing cryptocurrencies. Since users can place arbitrary computation in a script, miners in Ethereum have high incentive to skip verifying a block. For example in Figure 4, one can set n arbitrarily large to create a contract that requires, say, 30% of the computational power used in mining a block to verify its execution. That demand slows down the mining process of miners and incentivizes rational miners to skip verification in order to maintain their search speed in finding new blocks. Even though the remaining honest miners still account for the majority of the computational power in the network, their effective power is reduced by 30%. Thus, with high probability the rational miners will find more blocks and get a longer blockchain by skipping the verification of these transactions.

As we mentioned earlier, Ethereum introduced constraints on the rate of `gasLimit` variation. Unfortunately, however, we discuss in Appendix A why the `gasLimit` constraints are not a fool-proof mitigation against our attacks.

In Bitcoin, although the amount of work required to verify all the transactions is smaller, miners have non-zero incentive to skip verification as well. Indeed, on July 4 2015, a serious incident on the Bitcoin network was reported, wherein large pools extended blocks containing invalid transactions [27]. These pools mined on blocks without first verifying the block's transactions and caused a fork in the blockchain. As the verifier's dilemma describes, rational miners lack immediate economic incentives to verify transactions in newly mined blocks. The computational effort exhausted in verifying transactions detracts from the race to mine subsequent blocks, and it is possible that these pools skipped verification in order to gain computational advantage in the mining race.

[2] In Bitcoin, the protocol picks the blockchain which has more amount of work (the sum of the difficulties in blocks). However, it will be almost the same as comparing the lengths of both the blockchains.

4. INCENTIVIZING CORRECTNESS

In this section, we study and address the design drawbacks of the consensus protocol to prevent the aforementioned Attack 1 and Attack 2. Our goal is to incentivize miners to verify all transactions in each new block broadcast to them. Miners who deviate from the protocol should gain negligible advantage, and honest miners who verify all transactions should suffer negligible disadvantage from dishonest or malicious miners. Our consensus-based computation protocol below is not tied to any particular cryptocurrency, however we show how one might realize it in Ethereum in a way so as to achieve correct computations.

4.1 Consensus-based computation model

We define the consensus computation model, which formalizes the verification process of transactions/contracts in any consensus protocol, as follows.

Definition 1. A *consensus-based computation protocol* employed by a *consensus computer* involves three parties.

- Problem giver (G): who needs a solution for his particular problem and provides a function f which verifies the correctness of a solution.
- Prover (P): who submits a solution s to claim the reward from G.
- Verifiers (V): miners in the network execute $f(s)$ to decide whether s is a correct solution. In addition V always try to mine a new block, which requires W_{blk} work on average, in order to gain a reward.

In Bitcoin, the verification function f is rather simple. For the transaction in Figure 1, the problem that a sender (G) asks is to determine whether a receiver is the intended payee. The solution s that the receiver (P) needs to provide is his public key signed by his private key. The miners (V) will execute a function f defined in `scriptPubKey` to determine if s is correct and P can spend the received amount in that transaction. Miners all try to find new blocks to get reward as newly minted coins.

Let us denote by W_f the amount of work required to execute a verification function f. We define the advantage of rational miner as follows.

Definition 2. The *advantage for a rational miner* by skipping the verification of a transaction with verification function f is:

$$\mathrm{Adv}(f) = W_f - W_{df}$$

where W_{df} is the work required by deviating from the honest protocol.

Generally $W_{df} = O(1)$, which is the cost of picking a random result in $\{0, 1\}$ or even $W_{df} = 0$ if the miners just answer a constant value. Based on Definition 2, the advantage that a dishonest miner can get by skipping the verification process in one block is:

$$\mathrm{Adv}(\mathsf{Blk}) = \sum_{i+1}^{N} \mathrm{Adv}(f_i) = \sum_{i=1}^{N} W_{f_i} - O(1)$$

where N is the total number of transactions in a block Blk, and f_i is the verification function for the i-th transaction in Blk. Our threat model assumes an ε-rational miner as defined in Definition 3.

Definition 3. An *ε-rational miner* is one who honestly verifies the transactions in a block iff

$$\mathrm{Adv}(\mathsf{Blk}) \leq \varepsilon W_{\mathsf{blk}},$$

and deviates from it otherwise.

A ε-rational miner is incentivized to deviate from the honest protocol if the the work required to verify all transactions in a block is higher than a threshold $\varepsilon W_{\mathsf{blk}}$. Being dishonest helps him have a higher chance of finding the next blocks since others have to do a significant amount of work verifying transactions. On the other hand, if the work required is less, skipping the verification does not gain him much advantage (*e.g.*, lesser than other practical factors like network latency) to make his blockchain the longest one. Doing that may even risk his new block value if other miners detect that the previous block includes invalid transactions.

Incentivizing correct consensus computation. We incentivize miners to correctly execute f by limiting the amount of work required to verify all transactions in a block. Our goal is to provide an upper bound in the advantage that miners get by deviating from the honest protocol. This means honest miners are also guaranteed not to run long and expensive scripts while verifying the transactions. More specifically, we propose a new model defined as follows.

Definition 4. An *ε-consensus-based computation* protocol is a protocol in which the total amount of work that all verification functions require miners to do is *at most* $\varepsilon W_{\mathsf{blk}}$ per block.

A ε-consensus computer is a consensus computer that follows an ε-consensus-based computation. From Definition 3, our ε-consensus computer is incentive-compatible w.r.t. ε-rational. More specifically, it incentivizes miners to behave correctly, *i.e.*, honestly verify all transactions in a block.

4.2 Building an ε-consensus computer in Ethereum

One can estimate the value of ε to be the largest amount of "common good" work that the majority of miners find acceptable. This value, however, depends on several factors including the real net-worth of applications on the currency, the network properties, the incentive mechanism in the cryptocurrency, and individual miner's beliefs about the currency's value. Estimating ε is a separate and interesting research problem. Our next goal is to design a cryptocurrency network which supports ε-consensus computing for a specific pre-chosen value ε.

We next describe how to support ε-consensus computer based on Ethereum without requiring any major changes in its design. We further discuss which classes of computations can be run correctly on the existing cryptocurrency. While in general it is non-trivial to estimate the computation required by programs written in a Turing-complete language [28], the gas charged for a transaction is a reliable indicator of the amount of work required in that transaction script. To make our approach clearer, we define a gas function $G(x)$ as in Definition 5.

Definition 5. The *gas function* $G(x)$ determines the maximum *gas* amount that a program can require to do x amount of work.

Since Ethereum already specifies the gas value for each opcode in their design [23], computing $G(x)$ is relatively easy. Moreover, $G(x)$ can be computed once and used for all transactions. It only requires to update $G(x)$ again if the gas value is changed, or more opcodes are enabled.

We need only introduce a critical constraint on the transaction to make Ethereum an ε-consensus computer. That is, miners should only accept to verify the transaction that has `gasLimit` bounded by $G(\varepsilon W_{\mathsf{blk}}/N_0)$, where N_0 is the maximum number of transactions can be included in a block. Generally N_0 is fixed and represents the upper bound on the computational capacity of the network in one block time. We formally state our solution in Lemma 1.

Lemma 1 (Achieving ε-consensus computer in Ethereum)**.** *Given a specific ε value, one can construct an ε-consensus computer from Ethereum by limiting the* `gasLimit` *in every transaction by*

$$G\left(\frac{\varepsilon W_{\mathsf{blk}}}{N_0}\right).$$

Proof. Let us denote W_{tx} as:

$$W_{\mathsf{tx}} = \frac{\varepsilon W_{\mathsf{blk}}}{N_0}.$$

Since the `gasLimit` is at most $G(W_{\mathsf{tx}})$, W_{tx} is the upper bound on the amount of work required to verify a transaction. Thus, the work required in verifying all transactions in a block is no greater than $\varepsilon W_{\mathsf{blk}}$. By Definition 3, this an incentive-compatible strategy for ε-rational miners. $\square$

There are certainly classes of computations that require more than W_{tx} and even $\varepsilon W_{\mathsf{blk}}$ work to execute. While one can code such scripts in Ethereum, these puzzles fall outside the class of computations that our ε-consensus computer model can guarantee to execute and verify correctly.

4.3 Supporting more applications on an ε-consensus computer

We discuss two techniques for supporting puzzles which require greater than W_{tx} verification in Ethereum. One technique achieves correctness but sacrifices performance latency since it distributes the computation across multiple transactions and blocks that fit in the ε-consensus computer model. This technique effectively amortizes the verification cost across multiple transactions and blocks in the blockchain. The second technique, on the other hand, sacrifices only exactness and achieves probabilistic correctness.

4.3.1 Exact consensus computation

We introduce our first technique in the context of exact solutions. Here we simply split the execution of the script into smaller steps such that verifying each step requires at most W_{tx} work. Once all the steps have been run and verified as separate scripts, the outcome is the same as the result one would obtain from correctly executing the original script. Although the total verification work that the ε-consensus computer has to do is the same regardless of whether we use a single or multiple scripts, splitting the task guarantees a correct outcome because the ε-consensus computer correctly verifies smaller. We illustrate an alternative implementation for outsourcing matrix multiplication in n^2 steps where W_{tx} is $O(n)$ in Figure 5 (instead of 1 transaction where W_{tx} is $O(n^3)$ as in Figure 4).

The execution of the contract in Figure 5 works as follows. P first submits C to the contract as the solution for $A \times B$.

```
1  init:
2      ...
3      contract.storage[7] == 0 #no. of rounds
4  code:
5      ...
6      if msg.datasize==1:
7          contract.storage[8]=msg.data[0] #store C
8          contract.storage[9]=msg.sender #store prover
9          return ("Submitted C")
10     #verify the result
11     elif msg.datasize == 0:
12         C = contract.storage[8]
13         i=contract.storage[7]/n
14         j=contract.storage[7]%n
15         contract.storage[7] += 1
16         cell = sum([A[i][k] * B[k][j]
17                     for k in range(n)])
18         if cell != C[i][j]:
19             return ("Invalid result")
20     #after n^2 checks have passed
21     #send the reward to the prover
22     if contract.storage[7] == n*n:
23         send_reward()
24         ....
```

Figure 5: Matrix multiplication with $O(n)$ work per transaction on a consensus computer.

The verification happens across the next n^2 steps when P sends n^2 transactions to the contract. Each transaction verifies a particular and different element $C_{i,j}$ of the submitted result. A counter stores the number of passed checks (Lines 13–15).

Advantage across multiple transactions. A careful reader may be concerned that a rational miners might recognize the global connection between the n^2 transactions and skip verifying all of them to gain an advantage. However, the advantage from skipping the verification of transactions in the i-th block only helps the miners find the $(i+1)$-th block faster. In other words, one cannot “save” the “unspent” advantage in skipping the verification in one block and use that in the future since the competition for finding a new block “restarts” after every block. Essentially, we amortize the advantage of the computation across multiple transactions and blocks such that at any given block, the advantage of rational miners is bounded by $\varepsilon W_{\mathsf{blk}}$. Thus our approach does not break the incentive compatibility of the ε-consensus computer model.

Depending on the nature of the application, multiple steps can either happen in parallel or must be executed sequentially. For example in matrix multiplication, P can send simultaneously n^2 transactions. The reason is verifying $C_{i,j}$ does not rely on the correctness of $C_{i,j-1}$, and the order of verifying the two elements does not matter. On the other hand, if, say, $C_{i,j} = 2 \cdot C_{i,j-1}$, the prover would have to wait until $C_{i,j-1}$ gets verified before sending the next transaction which verifies $C_{i,j}$. In section 6, we illustrate these two scenarios with case studies.

4.3.2 Approximate consensus computation

Our second approach avoids the latency of sequential computation processes by employing probabilistic, light-weight verification. This approach will guarantee the correctness of the solution to a certain (adjustable) extent while only having to check a small portion of the solution.

Definition 6. Let us denote a problem $\mathsf{Z} : \{0,1\}^n \rightarrow \{0,1\}^m$, an input $x \in \{0,1\}^n$ and a claimed solution $y' \in$

$\{0,1\}^m$ of $\mathsf{Z}(x)$. We say Z is (δ, λ)-*approximate verifiable* if there exists a verification function f such that f accepts y' only if y' differs from $Z(x)$ in at most δ bits with probability greater than λ.

Our task is to encode f to ensure that if a solution y' is deemed correct only when y' does not differ much from the correct solution, *i.e.*, different in at most δ bits. Furthermore, in order to run f in a ε-consensus computer model, f should not require more than W_{tx} work to finish. The high level insight to encode such a f is that by randomly sampling and checking an entry of the output, there is some probability that we can detect if the output is incorrect at that sample. If we find an incorrect sample, we can conclude that the submitted solution is wrong and reject the solution. The more samples we take, the better the probability of catching an error. Otherwise, the solution is close to a correct one with an overwhelming probability greater than λ.

In Definition 6, we avoid defining too precisely the notion of "bits" and distance between two solutions since they vary slightly on specific encoding of the verification function f. One can translate "bits" as the positions in an array, or the pairs of elements, where the solution differs. In either case, the fraction of errors δ would change accordingly to the definition of "bit." For example, in the case of sorting, each "bit" would correspond to a pair (i, j) where $i < j$ and a bit in a solution y' differs from that in $\mathsf{Z}(x)$ if $y'[i] > y'[j]$.

We borrow the above sampling idea from the property testing literature which shows that in many practical instances one can determine whether a large object has a desired property by inspecting a small number of samples [29, 30]. For instance, property testing is a technique which allows a verifier to sample the output to decide whether an array is sorted. Property testing differs from verifiable computation, however, in that verifiable computation deals with general computations whereas property testing can only consider decision problems. Decision problems are not interesting for consensus verifiability because without doing any work a prover P can simultaneously post two solutions, "0" and "1," and one of these is guaranteed to be correct. Such an answer shifts the entire burden of computation onto the verifier unless the prover also provides some certificate which helps the verifier to check his answer more quickly. To use property testing in verifiable computation, we need to check the two following properties:

- *Property 1.* The provided solution differs in at most δ-bits from a solution that satisfies the property of the computation with high probability.
- *Property 2.* The provided solution is computed from the given input x with high probability.

Note that given a particular number of samples, the correctnesses that we can guarantee for *Property 1* and *Property 2* are different. For example, in the case of sorting a n-element array, n random samples may be sufficient to check whether the provided solution has the same elements as the given array with a $1 - \lambda = 99\%$ guarantee (*Property 2*). However, it may take more than n samples to fully check if the provided solution is sorted (*Property 1*). Thus, to achieve the overall 99% guarantee of correctness for both *Property 1* and *Property 2*, the number of samples one must take is the maximum of the numbers of samples to attain 99% guarantee of correctness for either of the checks.

5. IMPLEMENTATION

In this section we discuss the challenges while encoding a verification function f in our ε-consensus computer model. We further discuss techniques to address those challenges.

5.1 Challenges in implementation

The presence of contract and Turing-complete language in Ethereum enables a verifiable computing environment in which users can ask anyone to solve their problem and get the results verified by the network. We have established how to encode those verification computations f in Section 4.2 to work with our ε-consensus computer model. In fact, P can do the computation to arrive at the solution on his physical computer. G encodes the verification function f such that it takes P's solution and auxiliary data for the verification process. For example, if G asks P to sort an array, G can encode f to ask P to provide the sorted array (result) with the map between the result and the input. Our previous contract in Figure 5 is a concrete example where the matrix multiplication is outsourced to the network. We summarize the properties such a contract in our ε-consensus computer model can achieve.

1. *Correctness.* G receives correct results.
2. *No prior trust.* No pre-established trust between P and G is required.
3. *No interaction.* No interaction between P and G is required in order to verify the result.
4. *Efficiency.* G and P have to do only a modest amount of work.
5. *Fairness.* P is rewarded for a valid solution.

Properties 1–4 are immediate when we encode the verification function f in our ε-consensus computer model. Specifically, *correctness* is guaranteed since miners are incentivized to verify all computation. P and G do not need to trust or know each other, yet G cannot deviate after knowing the result. However, there are several challenges while implementing those verification function using smart contracts. In fact, simple smart contracts like the one in Figure 5 cannot guarantee *fairness* property due to one of the challenges that we describe below.

- **Insecure relay attack.** Once P finds a solution C, he broadcasts a transaction having C to his neighbors and to the whole network. However, since the solution is in the plaintext, a rational neighbor node may copy C, delay or even discard the prover's transaction and forge a new transaction to claim the reward.
- **How to randomly sample in Ethereum?** In order to verify computations via sampling, we need a mechanism for generating random numbers. We want to make sure that the random generator is unbiased in seed selection, and that the seed is unpredictable. If it is biased or predictable, the correctness property may be violated since P can submit a solution which is correct only in the "bits" that will be sampled. Ethereum does not natively support a random generator operator. We discuss our solution in Section 5.2.

  Another desirable property of a random generator is that it should be consistent across the network, *i.e.*, it should return the same sample set to everyone. Otherwise an honest miner M could verify that a solution

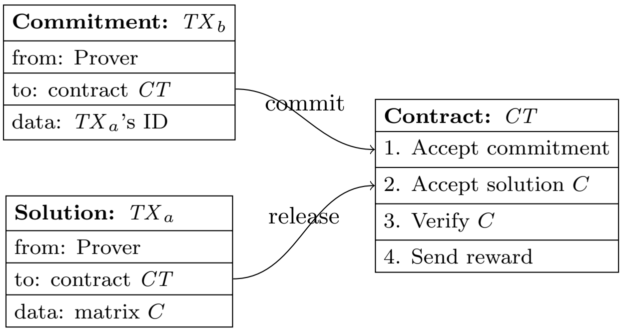

Figure 6: The commitment scheme used in our outsourced computation case studies. CT accepts the commitment in transaction TX_b before recieving solution in TX_a. That means P should see TX_b in the blockchain before he broadcasts TX_a to CT.

is correct with his samples but the other miners see it incorrect since they have a different sets of samples. Although M is honest, the other miners reject his block because with their sample sets M's transaction including his solution is invalid. It is unfair for M and problematic for the network as a whole since network consensus then becomes probabilistic.

One naïve approach to defeat the insecure relay attack above is for the prover P to encrypt and sign his solution since a key management system already exists in Ethereum. Unfortunately this does not work since every miner needs to access P's solution in order to verify the corresponding puzzle. In the next section, we devise a commitment scheme which helps us obtain *fairness* and discuss how to implement all the necessary components in our ε-consensus computer protocol.

5.2 Construction

We now resolve the bullet points raised above in Section 5.1.

Achieving fairness via a commitment scheme. A commitment scheme provides two important features. First, it ensures that other users cannot see, thus steal P's solution and claim the reward. Second, P once commits a solution cannot later alter it. This is to prevent some prover from submitting a fake answer to win the race with others (since only one prover is paid for correct solution), then spending more time computing the correct solution.

Our commitment scheme leverages the one-way hash function `SHA2`, which is already used in current cryptocurrencies to compute proof-of-work and transaction ID's. Specifically, we ask P to prepare a transaction TX_a that he includes his solution, and a transaction TX_b to commit TX_a to the contract. P first sends TX_b, which includes TX_a's ID, to the contract to say that he has the solution and will send that in the next transaction TX_a—*commit phase*. Once the contract accepts P's commitment, he sends TX_a and proceeds further as in without having the commitment—*release phase*. Our commitment scheme is shown in Figure 6.

Since an ID is computed by crytographically hashing the transaction data, other miners by observing TX_a's ID in TX_b are not able to construct TX_a to get C. Moreover, P cannot alter TX_a by a different solution since doing that will raise a conflict between the committed ID and the new one. In addition, once TX_b gets accepted and P broadcasts TX_a, the neighbors cannot claim the rewards with the solution observed in TX_a. This is because the contract is waiting for transaction TX_a which has been committed before.

Random sampling in Ethereum. Recall that for probabilistic verification, we need a psuedo random generator to randomly sample elements in a solution. Our key idea is to leverage some data in future blocks that miners do not know and cannot predict at the time of creating the transaction/contract. For instance, the randomness source can be the hash of the next block. Given a pseudo-random variable R as the next block hash, one can generate n other pseudo-random numbers simply by taking a hash as:

$$\texttt{SHA256}(R \,||\, i)$$

in which $1 \le i \le n$, and $||$ is a concatenation operator.

Since the information of a block is public and consistent in the network, all miners when run the above pseudo-random generator will get the same set of samples, thus achieving consistency. Further, the information of block is unknown before it is mined by miners, coupling with the randomness of `SHA256` function makes our pseudo-random generator fair.

6. CASE STUDIES

In this section, we exhibit several problems that can be solved by using our ε-consensus computer model. We are interested in the problems that require high computational resource to verify if are encoded naïvely without using our techniques discussed in Section 4.2. The purpose of those examples is to illustrate the practicality of our techniques, and also describe how to encode f for various δ-approximate verifable problems. Our examples consists of several problems in diverse domains such as Graph Theory, Linear Algebra, Number Theory and simple operations like matrix multiplication and sorting. In several of these cases we are able to easily verify whether a solution is correct. However, for several interesting problems verifying the correctness of a solution appears to be elusive. We try to circumvent this difficulty by taking a recourse in an approximate verifiability. We illustrate how one can encode the verification function f to employ the light-weight verification and approximately verify the correctness of several problems. For the convenience of readers, we list our case studies in Table 1. In this section, we consider the basic operations as basic arithmetic operations, *e.g.*, addition, multiplication and comparison over 32-bit integers, unless otherwise stated.

6.1 Exact computations

We show several applications that we can encode f to guarantee the correctness of a solution in an ε-consensus computer model. We also discuss the potential high latency due to the need of distributing the computation to multiple transactions.

6.1.1 GCD of Large Numbers

This example computes the exact GCD of two large numbers, each of size n bits. The usual way to solve to the *greatest common divisor* problem on a classical computer goes via the Euclidean algorithm which takes work proportional to $O[\log^3(n)]$. We show that by encoding the verification function differently, one can verify the result in quadratic time $O[\log^2(n)]$. More interestingly, this example requires only a single solution transaction — that is, guarantees exactness

Problems	Exact	Approx	# TXs	W_{tx}	#. Rounds
GCD	✓		$O(1)$	$O(1)$	1
Dot Product	✓		$O(n)$	$O(1)$	$O(\log n)$
Matrix Multiplication (Fig. 4)	✓		$O(1)$	$O(n^3)$	N/A
Matrix Multiplication (Fig. 5)	✓		$O(n^2)$	$O(n)$	N/A
Matrix Multiplication (Section 6.1)	✓		$O(n^3)$	$O(1)$	N/A
Matrix Multiplication		✓	$O(n)$	$O(1)$	$O(\log n)$
Sorting		✓	$O(1)$	$O(1)$	N/A
k-coloring		✓	$O(1)$	$O(1)$	N/A

Table 1: Case studies for outsourced computation in our ε-consensus computer model. W_{tx} represents the amount of work required in each transaction. *#. Rounds* is the number of rounds required to have a solution verified if the transactions have to send in order (N/A means there is no such order).

with no additional latency. Specifically, the algorithm is as below.

1. G posts two integers m and n.
2. P posts five integers a, b, x, y, and z.
3. V checks that:
   (a) $ax = m$, $bx = n$,
   (b) $|y| < b$, $|z| < a$, and
   (c) $ay + bz = 1$,
   (d) If all of these checks succeed, then V accepts P's solution. Otherwise V rejects P's solution.

We claim that if both checks above succeed, then $\gcd(m, n) = x$. If (a) is satisfied, then clearly x divides m and y divides n. To see that x is the smallest such number, we appeal to Bézout's Identity which tells us that a and b are relatively prime iff there exist y and z satisfying (b) and (c). If some integer greater than x divided both m and n, then by uniqueness of factorization, x would also divide that integer, forcing a and b to have a nontrivial common factor. Our verification uses 10 arithmetic operations over inputs of size $\log(n)$.

6.1.2 Dot product

We compute the exact dot product of two vectors

$$(a_1, \dots, a_n) \cdot (b_1, \dots, b_n) = a_1 b_1 + \cdots + a_n b_n$$

by way of a sequence of transactions on the consensus computer. The puzzle giver G need not perform a single basic operation, while the miners perform no more than 3 basic operations per transaction. Each transaction will permit us to reduce the number of additions in the running sum by one. The puzzle giver G will post $\lceil n/2 \rceil$ transactions to the first block, $\lceil n/4 \rceil$ transactions to the second block, $\lceil n/8 \rceil$ to the third, and so on for a total of at most n transactions. For simplicity of presentation, we will assume that G knows the quantity n, however this assumption is not necessary. One could modify the protocol below so that we iterate over all indices i for which a_i and b_i are both defined (with possibly one extra pair $a_{n+1} = b_{n+1} = 0$ added in case n is odd), and similarly for the partial sums in the subsequent stages.

1. For each $i \le \lceil n/2 \rceil$, G creates a puzzle transaction $T_{1,i}$ requesting the sum $a_{2i} b_{2i} + a_{2i+1} b_{2i+1}$.
2. For each i, P posts a number $s_{1,i}$ equal to the requested sum, in some permanent, public place.
3. For each i, V accepts P's solution iff $s_{1,i} = a_{2i} b_{2i} + a_{2i+1} b_{2i+1}$.

Subsequent stages k, for $k = 1$ up to the least k such that $2^k \ge n$ proceed similarly in a recursive way:

1. For each $i \le \lceil \lceil \lceil n/2 \rceil /2 \rceil \dots /2 \rceil$ (k 2's in this expression), G creates a puzzle transaction $T_{k,i}$ requesting the sum $s_{k-1,2i} + s_{k-1,2i+1}$.
2. For each i, P posts a number $s_{k,i}$ equal to the requested sum, in some permanent, public place.
3. V accepts P's solution iff $\forall i \; s_{k,i} = s_{k-1,2i} + s_{k-1,2i+1}$.

By induction on k, if for each i P's solution is correct, then $\sum_i s_{k,i}$ is equal to the desired dot product. In the final stage, we obtain a single value which equals the dot product, and so our multi-stage protocol succeeds. It is easy to verify that the work required in each transaction is $O(1)$ [3].

6.2 Verification with random sampling

As discussed earlier, some applications have solutions which are not easy or cheap to be verified. We show how to encode the verification function f for such applications to achieve probabilistic correctness with much smaller latency.

6.2.1 Approximate Sorting

We apply the direct sampling method so that with high probability the consensus will accept a correct solution but reject a solution with many errors. Let A denote the input array of $|A| = n$ elements. Let B denote the *claimed* output array of n elements and f denote a permutation on $\{1, \dots, n\}$, representing the *claimed* sorting order.

Definition 7 (δ-approximate sorting)**.** For $\delta \ge 0$, we say that an array B (together with a permutation f) is a δ-*approximate* solution to sorting array A if:

- at most δ fraction of elements of B are incorrectly mapped by f, i.e., $A[i] \ne B[f(i)]$, and
- at most δn^2 pairs of B are out of order.

We fix a $\delta > 0$ and let k denote the number of sample used by our protocol. The work required in the protocol

[3] In Figure 5, we showed how to compute the exact product of two matrices using n^2 transactions, each requiring $O(n)$ verification work. By applying the above mutli-transaction dot product contract n^2 times, one could also compute a matrix product via transactions each requiring only $O(1)$ work.

below will be directly proportional to k. The protocol below guarantees that if the output is correctly sorted then it is accepted with high probability and if the output is *not* δ-approximately sorted then it is rejected with high probability. Thus we get the guarantee that if an output is accepted then it must be δ-approximately sorted with high confidence. The confidence can be made arbitrarily close to 1 at the cost of increasing the sample size.

1. G posts an array A of n elements.
2. P posts an array B of n elements and f, a permutation on $\{1, \ldots, n\}$.
3. V chooses k random indices $p_1, \ldots, p_k \le |A|$ and k random pairs of indices $(q_1, r_1), \ldots, (q_k, r_k)$ and checks:
   (a) if: for all $i \le k$, $A[p_i] = B[f(p_i)]$, and
   (b) if: for all $i \le k$, if $q_i \le r_i$ then $B[q_i] \le B[r_i]$.
   (c) If both the checks (a) and (b) succeed then V accepts P's solution. Otherwise V rejects.

The verification requires k equality checks and k sorting comparisons.

Soundness of the protocol: Let us first assume that P's solution (B, f) is a correct solution to sorting A, i.e., B has no errors in the bijection f and no pairs are out of order. In this case, check (a) and (b) will not fail as there are no errors. Thus the protocol will accept every correct solution. □

Proof of approximate correctness: On the other hand if the array B differs from A in more than $\delta|A|$ places, then with probability at least δ, V will detect this error in (a). Similarly, if at least $\delta|A|^2$ pairs of elements in B are out of order, then with probability at least δ, the verifier V will detect this error in (b). The probability that (a) is satisfied for all i is less than $(1-\delta)^k$, and similarly the probability that (b) is satisfied for all i is less than $(1-\delta)^k$. Thus the probability that both of the checks succeed is at most the product $(1-\delta)^{2k}$. The verifier V fails to detect an error only when both checks (a) and (b) succeed. If we choose k large enough such that $(1-\delta)^{2k} < \lambda$ then we the protocol will reject any solution that is not δ-approximately sorted with probability greater than $1-\lambda$. □

6.2.2 Approximate Matrix Multiplication

We want to guarantee that if our verification function f accepts C then at least $1-\delta$ fraction of the entries of C match with $A \times B$ with high probability. The idea is to randomly pick a small number of entries of C and verify that they are computed correctly. For this check one may use the protocol for the Dot Product described in Section 6.1.2. By a calculation similar to the one for Approximate Sorting in Section 6.2.1, $\log\lambda/\log(1-\delta)$ samples suffice to ensure (δ, λ)-approximate verifiability.

6.2.3 Approximate 2-Coloring

A *2-coloring* of a graph is an assignment of one of two specified colors to each of its nodes. An edge is colored *properly* if the two endpoints of the edge are assigned different colors. Recall that a graph is 2-*colorable*, or *bipartite* if there a 2-coloring of its nodes such that every edge is properly colored. A 2-coloring is δ-*approximate* if at most δ fraction of the edges are *not* colored properly.

Using the sampling method, we design a protocol for the following decision problem: does a given graph A have a δ-approximate 2-coloring? Our protocol is inspired by the property testing algorithm for 2-coloring and its correctness relies on Szemeredi's Regularity Lemma [31, 32].

1. G posts a graph A.
2. P posts either posts:
   (a) "yes" and a 2-coloring of A, or
   (b) "no" and an array of
   $$k(\delta) = \frac{34 \ln^4(1/\delta) \ln\ln(1/\delta)}{\delta}$$
   nodes from A.
3. V checks:
   (a) If P answered "yes," then V chooses $k(\delta)$-random nodes $v_1, \ldots, v_{k(\delta)}$ from A.
      i. V accepts P's solution if the subgraph induced on $v_1, \ldots, v_{k(\delta)}$ is 2-colorable, and
      ii. V rejects otherwise.
   (b) If P answered "no," then
      i. V accepts P's solution if
         A. P's solution is an odd cycle, and
         B. all the edges of the odd cycle are present in A.
      ii. V rejects if either of the above conditions fails.

Soundness of the protocol: With probability greater than 1/2, a graph is *not* δ-approximate 2-colorable if and only if a random subset of $k(\delta)$ nodes contains an odd cycle [33]. Thus if the graph is not δ-approximate 2-colorable then we will detect this with probability at least $1-\lambda$ by repeating the above protocol $\lceil\log_2(1/\lambda)\rceil$ times. On the other hand, if the graph is 2-colorable then no subset of $k(\delta)$ nodes will contain an odd cycle. Hence our protocol will correctly accept the solution. Thus our protocol computes correctly with high probability both when the answer is "yes" or "no." □

Complexity of verification: Note that if the graph is not δ-approximate 2-colorable then indeed we have a constant size witness for this. Hence the "no" answer from the prover has a light-weight verification. It remains to show that the "yes" answer from the prover also has an easy verification. This can be achieved by a sampling method similar to the one used for sorting. If more than δ fraction of the edges are not properly colored by P's 2-coloring, then we will catch a violated edge with high probability using $s = k(\delta) \cdot \lceil\log_2(1/\lambda)\rceil$ samples.

In the "yes" instance, V does s basic operations to choose the random nodes and s^2 comparisons to check that the subgraph inherits a 2-coloring. In the "no" instance, s basic operations are used to check whether P gave an odd cycle, and s basic operations to check it's presence in A for a total of $2s$ basic operations. □

Finally, we remark without proof that our protocol can in fact be modified to verify approximate c-coloring as well for any constant c using [33]; see also [32].

7. RELATED WORK

Consensus protocol. Since the Nakamoto consensus protocol first appeared [1], numerous works have investigated alternative consensus algorithms [34, 35, 36, 37]. The problem we look at in this work is independent of the underlying consensus algorithm used in the network as the *verifier's dilemma* arises in any cryptocurrency that has high block's verification cost.

Incentive compatibility in cryptocurrency. Apart from verifying blocks and transactions, mining new blocks is one major activity in cryptocurrencies. Block mining requires a huge computational resource, thus miners often join hands to mine together in a *pool*. Several previous works also study the incentive compatibility of Bitcoin mining [38, 39, 40, 41]. For example, in [38, 40], the authors prove that pooled mining protocol is not incentive compatible by showing that miners and pools are susceptible to a block withholding attack. Our work also studies the incentive compatibility in cryptocurrency, but via the lens of the verification activity. We show that in a network which allows Turing complete scripts, miners are vulnerable to a new class of attacks.

Security analysis of Bitcoin protocol. A recent paper by Gary *et al.* models and analyses security of the Bitcoin protocol [42]. Gary *et al.* prove that Bitcoin can achieve all ideal properties, *e.g.*, common prefix, chain quality, only if f is negligible, where f is the number of proof-of-work solutions that miners can generate while a block is broadcast. We find that our finding follows closely to their result. In fact, the parameter f in [42] typically depends on the parameter ε in our paper. The present work showed that when ε is sufficiently large, the verifier's dilemma exists. Our solution is to keep ε small, which is equivalent to keeping f close to 0.

Verifiable computation. Our proposed outsourced computation scheme on an ε-consensus computer is the first system to achieve all the ideal properties mentioned in Section 5. A popular line of work uses classical computers for verification that involves prior trust establishment to guarantee that G does not deviate after knowing the solution [8, 9, 10, 11, 12, 13, 14]. A key difference is in the assumptions and goals. Our goal is not to produce explicit proofs of correctness, rather to enable an incentive structure where cheating provides no intrinsic advantage over following the protocol honest. Secondly, our techniques are specific to certain algorithms, whereas verifiable computation enables proving correctness of arbitrary circuits. Moreover, verifiable computation can be and have been applied in our setting, since recent schemes have constant verification effort [43]. However, such schemes impose impractically high overheads on provers and require expensive inial key setup between all miners in the protocol.

Since the rise of Bitcoin, a new line of work on repurposing the blockchain for other applications has been initiated [19, 21, 16, 20]. For example, in [19, 21], Kumaresan *et al.* studied how to run several applications, including verifiable computation, on Bitcoin. Their technique is fairly complicated and relies on the assumption that all the computations done by consensus protocol will be correct. As we have shown, with expressive scripting languages (c.f. Ethereum), one can achieve what these previous works have done in Bitcoin with a single concise *smart contract*. However, as we pointed out in this paper, miners have incentive to deviate from the honest protocol in a new Turing-complete cryptocurrency. Thus, techniques used in [20, 21] may not guarantee the correctness of the computation on some ε-consensus computer having small ε. Furthermore, our technique differs from theirs by leveraging a new cryptocurrency design and adapting property testing techniques within the verifiable computation domain.

8. CONCLUSION

In this paper, we introduce a *verifier's dilemma* demonstrating that honest miners are vulnerable to attacks in cryptocurrencies where verifying transactions per block requires significant computational resources. We formalize the security security model to study the incentive structure and attacks which affect the correctness of computations performed on a consensus computer. Finally, we discuss how to implement our ε-consensus computer in Ethereum with various trade-offs in latency and accuracy. We consider it an interesting open problem to determine whether one can incentivize robust computations to execute correctly on a consensus computer by modifying its underlying consensus mechanism.

9. ACKNOWLEDGMENT

We thank Frank Stephan, Shruti Tople, Pratik Soni, Ratul Saha, Virgil Griffith and Vitalik Buterin for useful discussions and feedback on the early version of the paper. This work is supported by the Ministry of Education, Singapore under under Grant No. R-252-000-560-112. All opinions expressed in this work are solely those of the authors.

10. REFERENCES

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *bitcoin.org*, 2009.

[2] Karl J. O'Dwyer and David Malone. Bitcoin mining and its energy footprint. In *Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014). 25th IET*, pages 280–285, June 2014.

[3] Rob Halford. Gridcoin: Crypto-currency using berkeley open infrastructure network computing grid as a proof of work. http://www.gridcoin.us/images/gridcoin-white-paper.pdf, May 2014.

[4] Use case for factom: The world's first blockchain operating system (bos). http://kencode.de/projects/ePlug/Factom-Linux-Whitepaper.pdf, Feb 2015.

[5] Ethereum Foundation. Ethereum's white paper. https://github.com/ethereum/wiki/wiki/White-Paper, 2014.

[6] blockchain.info. Bitcoin average number of transactions per block. https://blockchain.info/charts/n-transactions-per-block.

[7] Gavin Andresen. Why increasing the max block size is urgent. http://gavinandresen.ninja/why-increasing-the-max-block-size-is-urgent, May 2015.

[8] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 71–89, 2013.

[9] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming

interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 90–112, 2012.

[10] Srinath T. V. Setty, Richard McPherson, Andrew J. Blumberg, and Michael Walfish. Making argument systems for outsourced computation practical (sometimes). In *19th Annual Network and Distributed System Security Symposium, NDSS*, pages 253–268, 2012.

[11] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. Cryptology ePrint Archive, Report 2013/507, 2013. http://eprint.iacr.org/.

[12] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 341–357, 2013.

[13] Srinath Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, 2012.

[14] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 238–252, 2013.

[15] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Naryanan, Joshua A. Kroll, and Edward W. Felten. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *IEEE Security and Privacy 2015*, May 2015.

[16] A Miller, A Juels, E Shi, B Parno, and J Katz. Permacoin: Repurposing bitcoin work for long-term data preservation. *IEEE Security and Privacy*, 2014.

[17] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. *IEEE Symposium on Security and Privacy*, 2013.

[18] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, 2014.

[19] Ranjit Kumaresan and Iddo Bentov. How to use bitcoin to incentivize correct computations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014*, pages 30–41, 2014.

[20] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 443–458. IEEE Computer Society, 2014.

[21] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 421–439, 2014.

[22] Nick Szabo. The idea of smart contracts. http://szabo.best.vwh.net/smart_contracts_idea.html, 1997.

[23] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. http://gavwood.com/paper.pdf, 2014.

[24] Andrew Miller, James Litton, Andrew Pachulski, Neil Spring Neal Gupta, Dave Levin, and Bobby Bhattacharjee. Discovering Bitcoin's public topology and influential nodes. http://cs.umd.edu/projects/coinscope/coinscope.pdf, May 2015.

[25] US-CERT/NIST. Resource exhaustion attack in bitcoin. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-2292, 2013.

[26] Sergio Demian Lerner. New bitcoin vulnerability: A transaction that takes at least 3 minutes to be verified by a peer. https://bitcointalk.org/?topic=140078, 2013.

[27] Some miners generating invalid blocks. https://bitcoin.org/en/alert/2015-07-04-spv-mining, July 2015.

[28] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Proving program termination. *Commun. ACM*, 54(5):88–98, May 2011.

[29] Dana Ron. Property testing (a tutorial). http://www.eng.tau.ac.il/~danar/Public-pdf/tut.pdf, 2000.

[30] Oded Goldreich. Combinatorial property testing (a survey). *Electronic Colloquium on Computational Complexity (ECCC)*, 4(56), 1997.

[31] Endre Szemerédi. Regular partitions of graphs. In *Problèmes combinatoires et théorie des graphes (Colloq. Internat. CNRS, Univ. Orsay, Orsay, 1976)*, volume 260 of *Colloq. Internat. CNRS*, pages 399–401. 1978.

[32] Noga Alon and Asaf Shapira. Every monotone graph property is testable. *SIAM Journal on Computing*, 38(2):505–522, 2008.

[33] Noga Alon and Michael Krivelevich. Testing k-colorability. *SIAM Journal on Discrete Mathematics*, 15(2):211–227, 2002.

[34] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, 1999.

[35] David Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. https://www.stellar.org/papers/stellar-consensus-protocol.pdf, April 2015.

[36] Kwon Jae. Tendermint: Consensus without mining. http://jaekwon.com/2014/05/11/tendermint/, May 2014.

[37] King Sunny and Nadal Scott. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. http://peercoin.net/assets/paper/peercoin-paper.pdf, August 2012.

[38] Loi Luu, Ratul Saha, Inian Parameshwaran, Prateek Saxena, and Aquinas Hobor. On power splitting games in distributed computation: The case of bitcoin

pooled mining. *The 28th IEEE Computer Security Foundations Symposium*, July 2015.

[39] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *arXiv preprint arXiv:1311.0243*, 2013.

[40] Ittay Eyal. The miner's dilemma. In *The 36th IEEE Symposium on Security and Privacy*, SP '15, May 2015.

[41] Joshua A. Kroll, Ian C. Davey, and Edward W. Felten. The economics of Bitcoin mining, or Bitcoin in the presence of adversaries. In *Workshop on the Economics of Information Security*, June 2013.

[42] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology - EUROCRYPT 2015*, pages 281–310, 2015.

[43] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. Cryptology ePrint Archive, Report 2015/675, 2015. http://eprint.iacr.org/.

[44] Response from founder of ethereum about our paper. https://www.reddit.com/r/ethereum/comments/3fcw0i/verifiers_dilemma_renders_ethereum_nonincentive/, July 2015.

APPENDIX

A. DISCUSSION ON GASLIMIT IN ETHEREUM

We explain why `gasLimit` does not help Ethereum prevent our DoS attack completely. The current design allows the miners to set the `gasLimit` for the next block once they find a block. However, the `gasLimit` of the next block cannot vary more than a fraction (2^{-10}) of the current `gasLimit`. We learned this from our private communication with the founder of Ethereum. The constraint seems to mitigate our attack, however, we explain why the `gasLimit` can reach to a high value that makes the resource exhaustion attack feasible.

In practice, miners have different views of what `gasLimit` value is acceptable because of various reasons. For instance, one may be willing to always verify a new block regardless of its `gasLimit` value because they have more resources or they simply value the advantages of having a high `gasLimit` value than the disadvantages. Thus, each miner will decide to only reduce block's `gasLimit` at a different threshold $\mathcal{G}_i$. Suppose that our DoS attack requires `gasLimit` to be at least $\mathcal{G}_0$ to be practical. In a scenario where more than 50% of computational power consider $\mathcal{G}_0$ is still within their $\mathcal{G}_i$, all miners are under our denial-of-service attack. On the other hand, if the majority of miners have their $\mathcal{G}_i$ less than $\mathcal{G}_0$, `gasLimit` can successfully block our attack.

For the completeness of our argument, rational users in the network have the following incentives to extend `gasLimit` value.

- Higher `gasLimit` value means higher transaction fee that miners can collect from a block.
- A block with a higher `gasLimit` can support more applications, specifically the application that requires more gas to run. Thus, the value of the network and its underlying currency is more valuable, which benefits directly the miners.
- As more and more applications are built on top of Ethereum, the `gasLimit` has to be increased correspondingly in order to improve the throughput of the network to support those applications. It is not practical to wait for, say, ten blocks to see a transaction gets included in the blockchain due to the small throughput.

There are also reasons that miners decide to reduce `gasLimit`. The Ethereum founder mentions about those reasons in a public post which discusses our work [44].

- Miners realize that high `gasLimit` value may cause DoS attack as we described.
- Transaction fee is increased because low `gasLimit` means only a limited number of transactions can be included in a block.

In conclusion, `gasLimit` does not completely block our DoS attack, *i.e.*, resource exhaustion attack, in Ethereum.